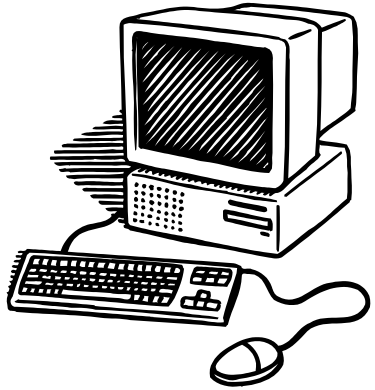
スパコンプログラマの日常
～スパコンを使うのって難しいの？～

東京大学物性研究所
物質設計評価施設
渡辺宙志

東大物性研マスコット
「物性犬」

スパコンとスパコンプログラマ



スパコン

＝スーパーコンピュータ→すごいコンピュータ

→コンピュータはソフトウェアがなければただの箱

　スパコンも例外では無い

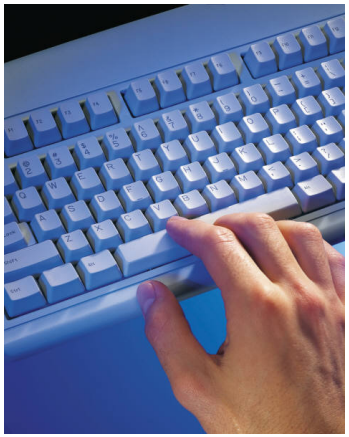
プログラマ

＝プログラムを組む人

→プログラムをビルドするとソフトウェアに

「京」コンピュータは世の中に一つしかない

→「京」用のソフトウェアは売ってないので作るしか無い



スパコンプログラマ

＝スパコンのためのプログラムを組む人

＝すごいコンピュータのためのプログラマ

＝すごいプログラマ？

スパコンにおけるプログラミングとはどんなものか？

スパコンプログラマは何をしている人なのか？

自己紹介 (1/2)　私とプログラム

プログラム歴

小学校	PC-9801FでN88-BASICを触る
中学校～高校	Quick BASIC + アセンブラでMS-DOSのゲームを作成
大学～	Borland C++ BuilderでWindowsのフリーソフト作成
大学院	数値シミュレーション、単純並列計算
名古屋大学情報科学研究科	ActionScriptを覚えてFlashをいくつか作成
東京大学情報基盤センター	MPIによる非自明並列計算
東京大学物性研究所	大規模分子動力学法で研究

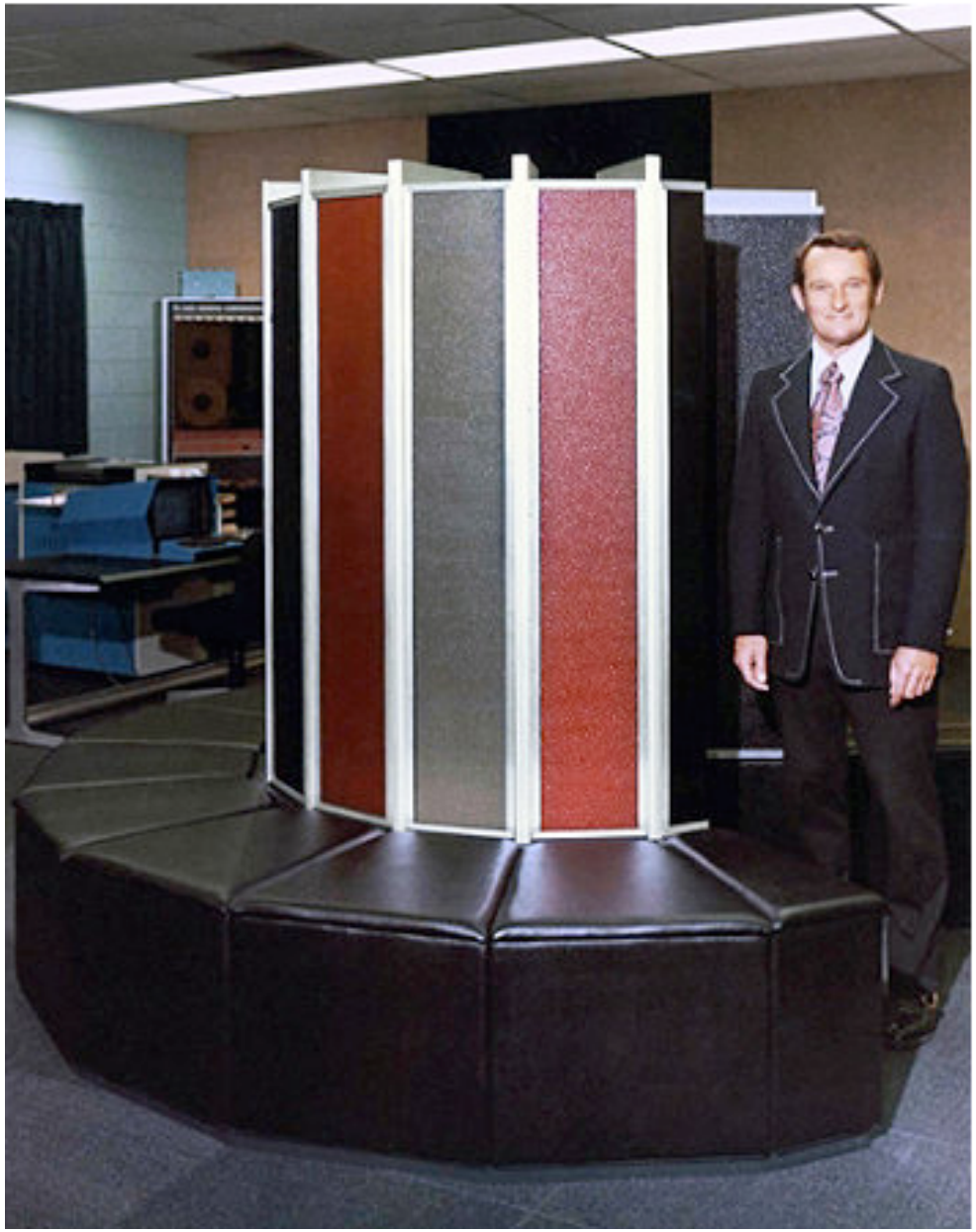
研究以外で作ったもの

MS-DOS	ドラクエ型RPG (文化祭用：サブプログラム、BGM、原画) 対戦アクションゲーム (コンテスト用：サブプログラム、BGM、原画) ダンジョンRPG (卒業制作：サブプログラム、BGM)
Windows	対戦アクションゲーム (MS-DOSで作ったゲームの移植) パズルゲーム2本 (プログラム、BGM) スクリプト言語の統合開発環境、簡易エディタ 迷路作成プログラム ← ヤンメガの裏表紙 量子計算シミュレータ ← 未踏ソフトウェア
Flash	クリックアクションミニゲーム パズルゲーム 交通流シミュレータや集団おにごっこシミュレータ

自己紹介 (2/2)　私とスパコン

1976年生まれ (38歳)
2001年 (D1): 物性研 SGI 2800/384
2002年 (D2): 筑波大 CP-PACS
2003年 (D3): JAMSTEC 地球シミュレータ
2004年 名古屋大学情報科学研究科へ助教として赴任
2005年 物性研 SGI Altix 3700
2008年 東京大学情報基盤センターへ特任講師として赴任
2009年 九州大学 SR16000 (42ノード) 25.3TFLOPS
2010年 核融合研究所 SR16000 (128ノード) 77TFLOPS
2010年 物性研究所へ助教として赴任
2010年 物性研 SGI Altix 8400EX
2012年 情報基盤センター FX10 (4800ノード) 1PFLOPS
2014年 理研「京」10PFLOPS

Cray-1とシーモア・クレイ

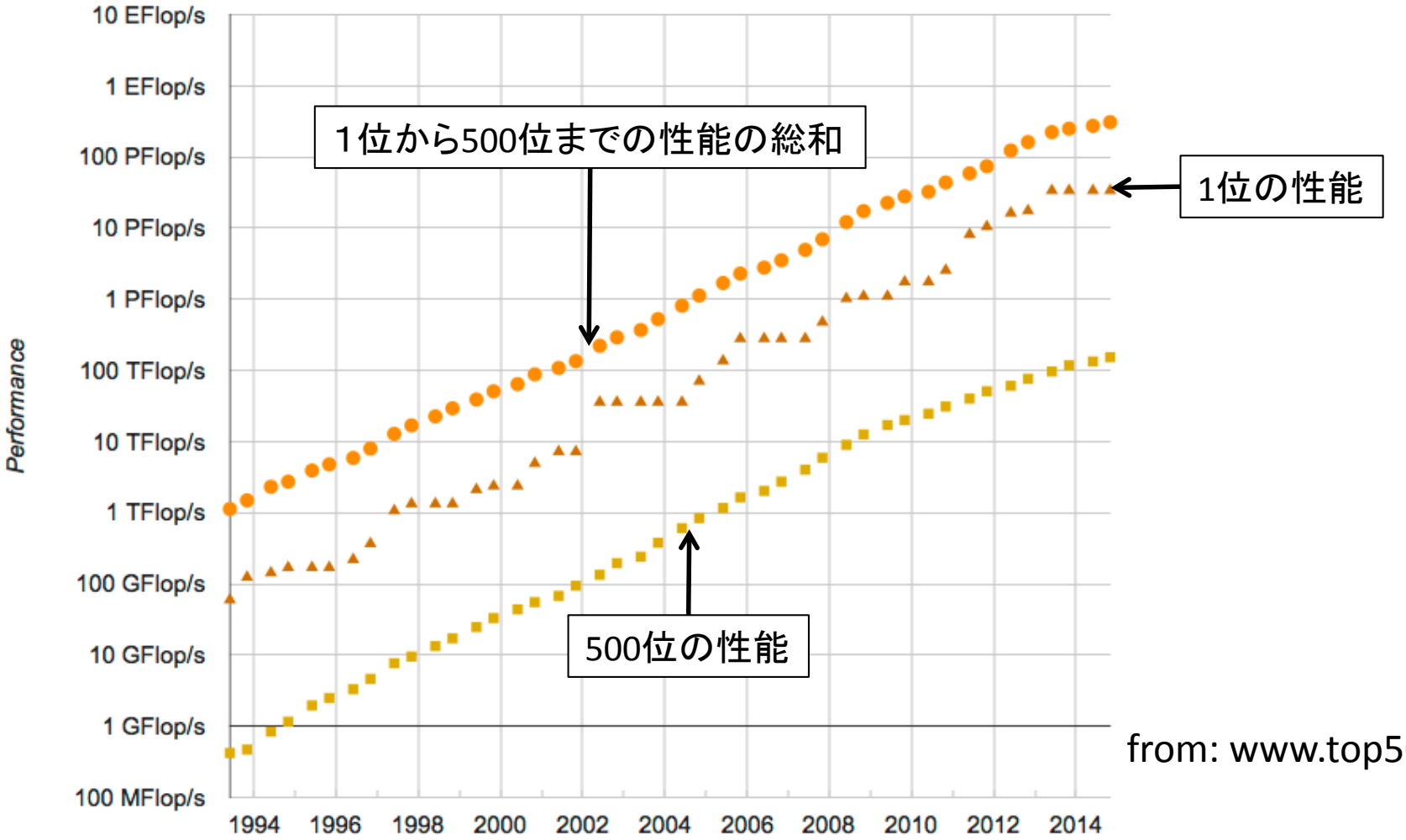


from: Computer History Museum
http://www.computerhistory.org/revolution/supercomputers/10/7/3

1976年出荷
ベクトル型スパコン
理論ピーク性能160MFLOPS

「スパコンの父」クレイ

計算機の性能の向上



from: www.top500.org

年率およそ184%で指数関数的に性能向上 (6年で約40倍)
性能向上に従って複雑化するアーキテクチャ
最新のアーキテクチャに乗り遅れたプログラムは、6年で40倍遅くなる
→ これをなんとかするのがスパコンプログラマ

気泡生成シミュレーション

工学応用上「泡」は厄介者

冷却システムで気泡発生→熱交換効率低下
スクリュー周りで気泡発生→騒音や腐食

➡ 泡を理解/制御したい

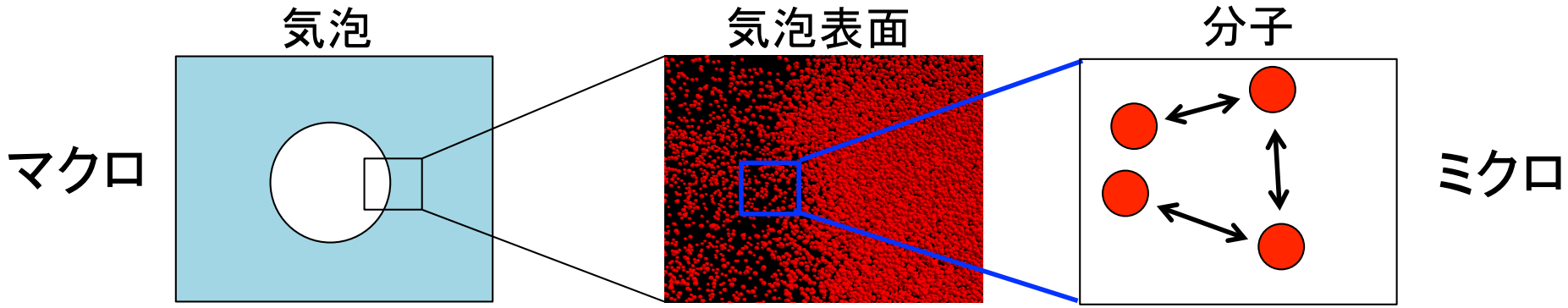
お湯を沸かす
(加熱による発泡)

from Wikipedia

スクリューまわりの気泡
(減圧による発泡)

from Wikipedia

泡の発生の研究は難しい
(ナノメートル程度の相互作用がミリ～センチメートル程度の現象を支配)



泡の発生・成長メカニズムを分子レベルから明らかにしたい

➡ スパコンで泡の発生をシミュレーション (スパコンプログラミング)

スパコンプログラマのお仕事

並列プログラミング

スパコンは多くのCPUから構成されている
→ スパコンを使うためには並列プログラミングが必須
ひとくちに「並列化」といっても、実は以下の三種類が存在
・ノード間並列 (分散メモリ、プロセス並列)
・ノード内並列 (共有メモリ、スレッド並列)
・SIMD (CPUコア内、ベクトル演算)

チューニング

スパコンは貴重なリソース → なるべくプログラムを高速化

例) 100億円のスパコンで走るプログラムを全て10%高速化したら10億円儲けたのと同じ

ある種のアーキテクチャではチューニングの効果が大きい
→ 計算機の仕組みを理解してプログラムを組むことが重要

プログラミング言語

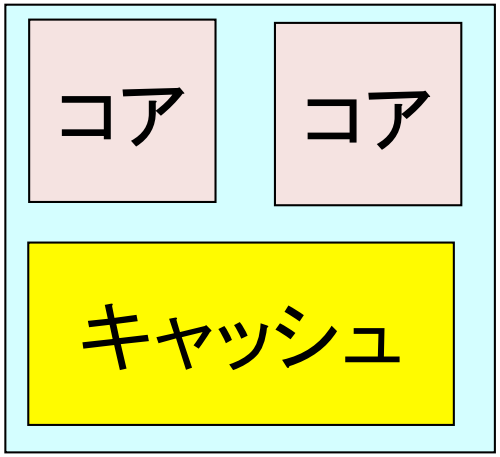
使用言語は事実上FORTRANかC++の二択

計算機の仕組み (1/3)

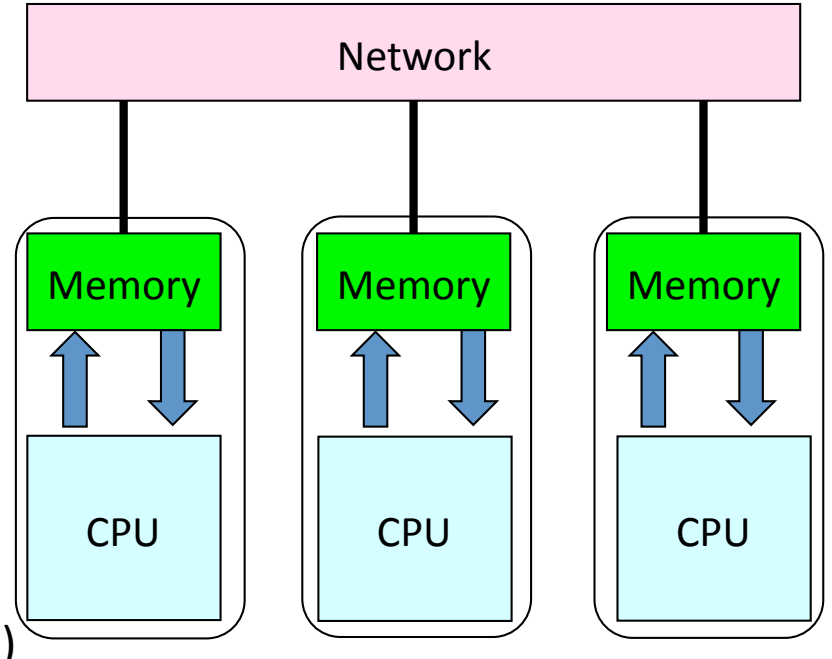
スパコンの性能を引き出すには、その仕組みを理解する必要がある

CPU=CPUコア + キャッシュ
ノード＝CPU＋メモリ
スパコン= たくさんのノードをネットワークでつないだもの

CPU



スパコン



一般的なCPUは、4コア～16コア程度 (「京」は8コア)
ノード構成は2CPUから4CPU程度 (「京」は1CPU)
典型的なスパコンは100～1万ノード程度(「京」は8万ノード)

スパコンのCPUはパソコンとあまり変わらない
どこがスーパーか？→ ネットワークと信頼性

例：京におけるLinpackの測定 88128CPU(705024コア)*29時間28分= 2374コア年
→ CPUコアとして2万年保証が必要

計算機の仕組み (2/3)

CPUとは

CPU (Central Processing Unit、中央演算処理装置)は、コンピュータの「頭脳」
様々な種類があり、各々のCPUごとに得意/不得意や、使い方の「クセ」がある。

ゲーム機とスパコンでの採用例

第五世代	SS	SH-2
	PS	R3000A (MIPS)
	N64	VR4300 (MIPS)
第六世代	DC	SH-4
	PS2	MIPS (Emotion Engine)
	GC	IBM PowerPC カスタム (Gekko)
	Xbox	Intel Celeron (Pentium III ベース)
第七世代	Wii	IBM PowerPC カスタム
	Xbox 360	IBM PowerPC カスタム
	PS3	IBM Cell 3.2
第八世代	PS4	AMD Jaguar

物性研 SGI Origin 2800 (MIPS)

KEK Blue Gene/Q (PowerPC)
via http://scwww.kek.jp/

LANL Roadrunner (Cell)
via http://ja.wikipedia.org/wiki/Roadrunner

計算機の仕組み (3/3)

ノード (国内)

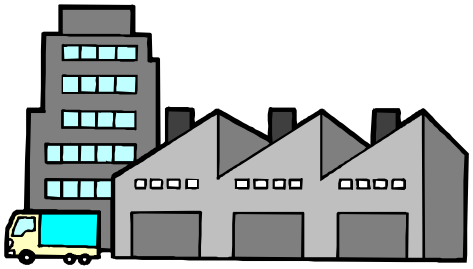
工場 (CPUコア)

物置
(キャッシュ)

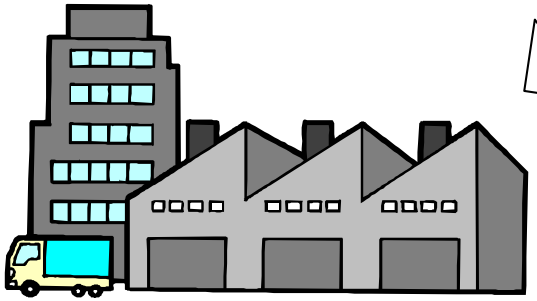
国内流通
(メモリ転送)

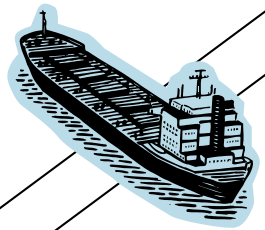
ボトルネック

倉庫(メモリ)



別の国の倉庫
(別のノードのメモリ)



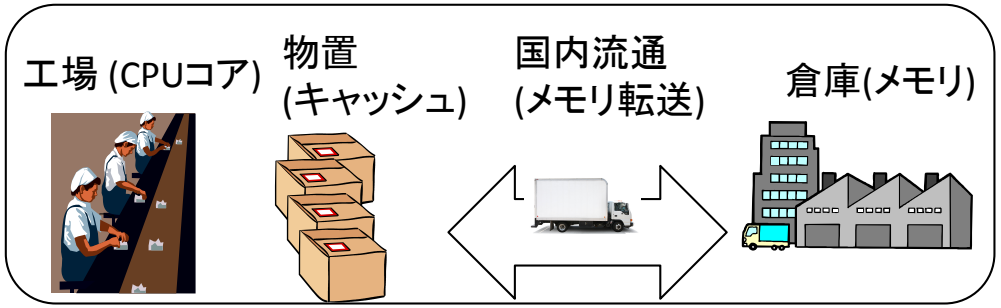


貿易
(ノード間通信)

メモリ転送の問題

CPUの仕事

- メモリからデータを取ってくる
- 計算する(ほとんど四則演算)
- 結果をメモリに書き戻す



レイテンシとバンド幅

レイテンシ：発注してからデータが届くまでの時間 (トラックの速度)

必要な材料が物置になかった→倉庫に発注
材料が届くまで工場は暇になる (およそ数百～千サイクル)

バンド幅　：定常的なデータの転送量 (道路の車線数)

現在の典型的なCPUでは、データを2個持ってきて1個書き戻す
間に50回くらい計算できる能力がある
＝工場の能力に比べて道路の車線数が少なすぎる

いかに物置(キャッシュ)を活用するか
無駄に計算してでも、データ転送量を減らす

命令パイプライン (1/2)

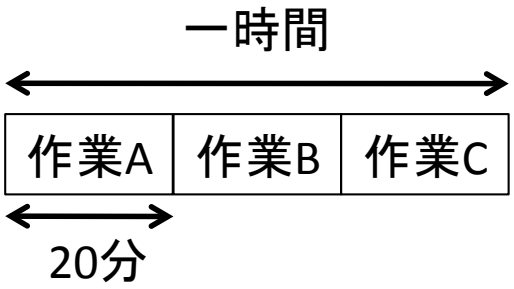
非パイプライン方式

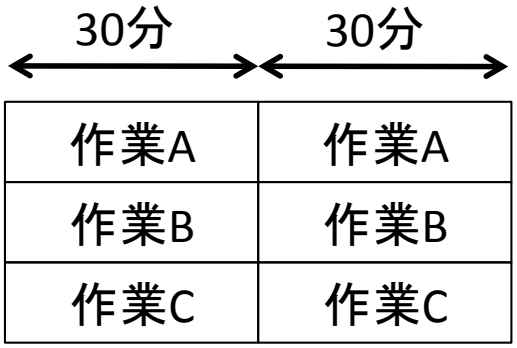
工場で熟練した職人が製品を作る
職人の手さばきが速くなるほど速く製品ができるが、限界がある

パイプライン方式

工程を簡単な作業にわけ、ベルトコンベア式に
作業員は自分のパートしかしない

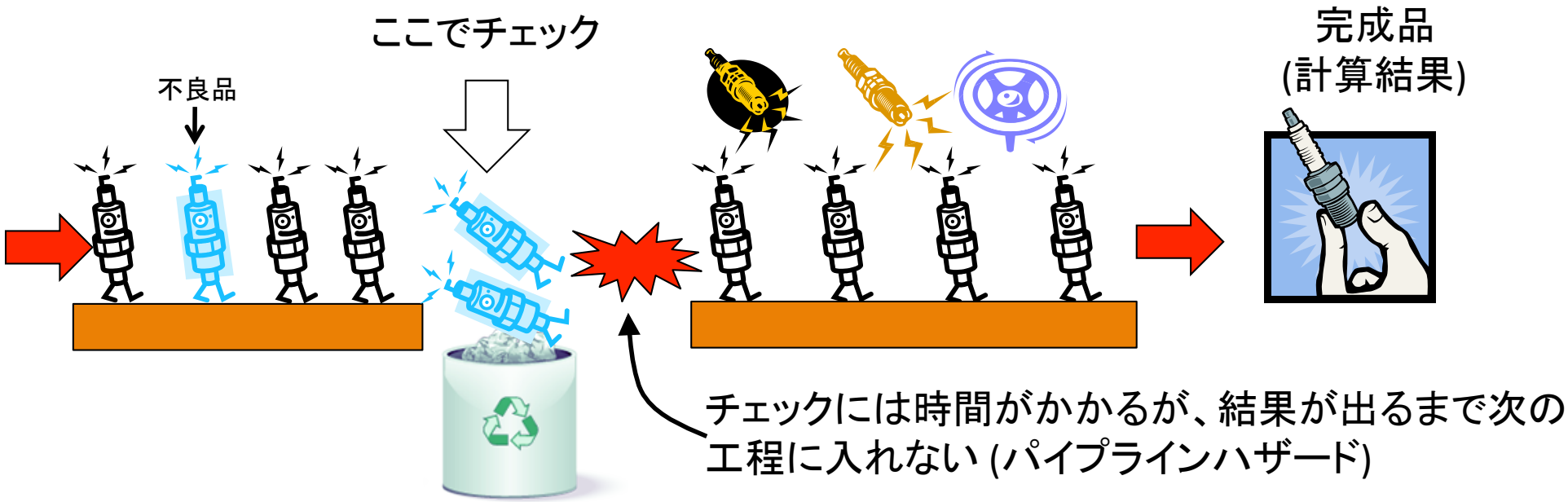
職人一人で作ると1時間かかる作業を3つにわける
ベルトコンベアに部品を流し、各パートを作業員に任せる
職人がやると20分でできる仕事が、作業員だと30分に
しかし、三人が同時に作業できるので、一つの製品が30分で完成 (2倍の高速化)

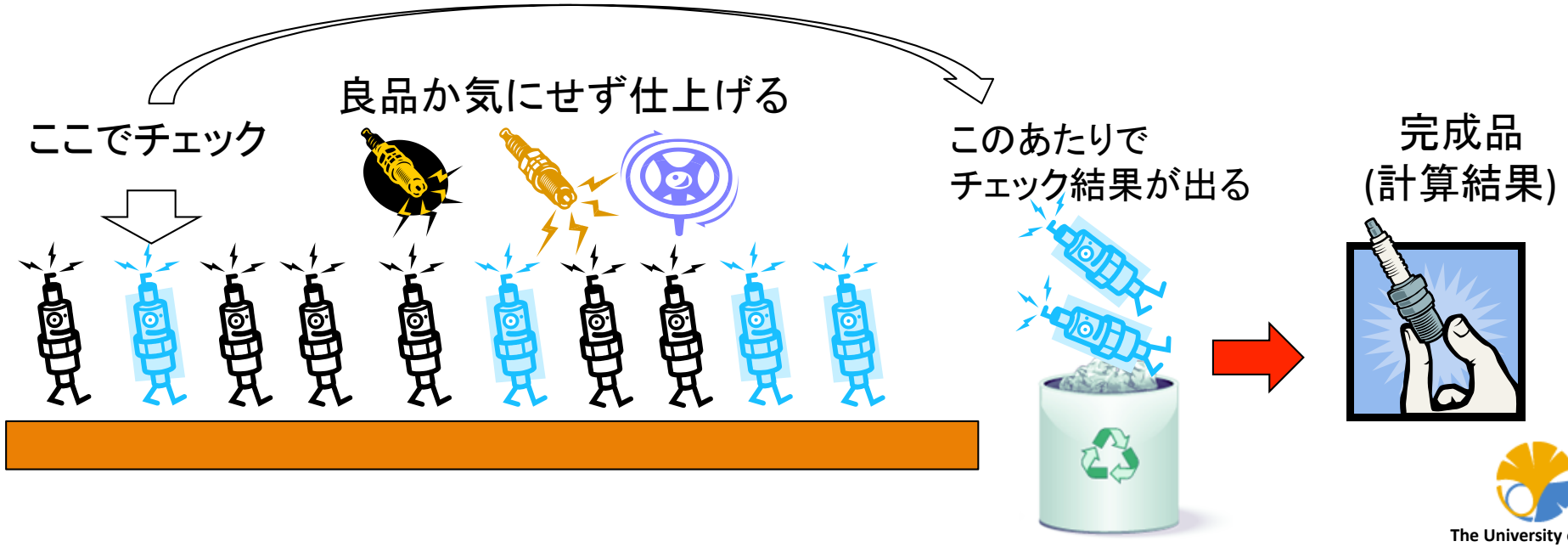




命令パイプライン (2/2)

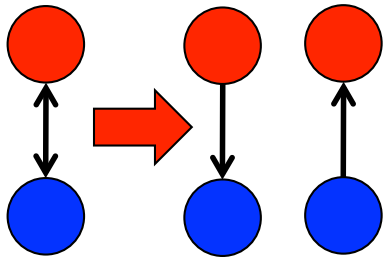
製品のチェックをして、良品だけ最後の仕上げをしたい





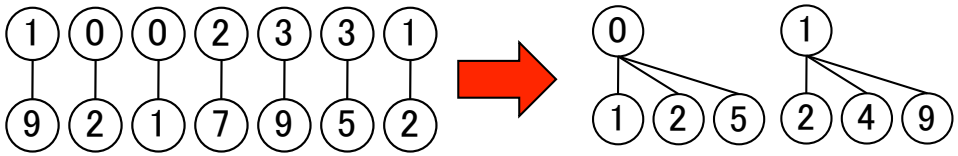
チューニング例

作用反作用無視



同じものを二回計算
メモリ転送量が半分に

相互作用粒子ソート



同じ粒子と相互作用する粒子をまとめる
メモリ転送量が半分に (レジスタ活用)

条件分岐削除

1. 粒子の距離を計算
2. ある程度以上遠ければ次のペアへ
3. 粒子間の力を計算
4. 速度を更新
5. 次のペアへ

1. 粒子の距離を計算
2. 粒子間の力を計算
3. もし距離が遠ければ力をゼロに上書き
4. 速度を更新
5. 次のペアへ

※実際にはソフトウェアパイプライニングというテクニックで、予め距離を計算し、判定に間に合わせる

除算削除

A1 = 1/C1
A2 = 1/C2

D = 1/(C1*C2)
A1 = D*C2
A2 = D*C1

除算二回を、除算1回乗算3回に変換
除算が遅いアーキテクチャで有効

その他細かいチューニング

除算のSIMD化のため、低精度逆数近似命令(frcpd)と精度補正＋ループアンロール＋手でソフトウェアパイプライニング

実際のコード例

力の計算ルーチン

```
for (int i = 0; i < pn; i++) {
  v2df qxi(q[i][X], q[i][X]);
  v2df qyi(q[i][Y], q[i][Y]);
  v2df qzi(q[i][Z], q[i][Z]);
  const int kp = key_pointer[i];
  const int np = number_of_partners[i];
  v2df vxiv2(0.0, 0.0), vyiv2(0.0, 0.0), vziv2(0.0, 0.0);
  int lj = kp;
  for (; lj < kp + np - 3; lj += 4) {
    int ja = sorted_list[lj];
    int jb = sorted_list[lj + 1];
    int jc = sorted_list[lj + 2];
    int jd = sorted_list[lj + 3];
    v2df dxa(q[ja][X], q[jb][X]);
    v2df dya(q[ja][Y], q[jb][Y]);
    v2df dza(q[ja][Z], q[jb][Z]);
    v2df dxc(q[jc][X], q[jd][X]);
    v2df dyc(q[jc][Y], q[jd][Y]);
    v2df dzc(q[jc][Z], q[jd][Z]);
    dxa -= qxi;
    dxc -= qxi;
    dya -= qyi;
    dyc -= qyi;
    dza -= qzi;
    dzc -= qzi;
    v2df RRa = dxa * dxa + dya * dya + dza * dza;
    v2df RRc = dxc * dxc + dyc * dyc + dzc * dzc;
    v2df _R2a = RRa.inv();
    v2df _R2c = RRc.inv();
    v2df mask_a = (RRa * _Rcf2v2).floor();
    v2df mask_c = (RRc * _Rcf2v2).floor();
    v2df Aa = v2df(8 * 48 * dt, 8 * 48 * dt) - v2df(7 * 48 * dt, 7 * 48 * dt) * RRa * _R2a;
    v2df Ac = v2df(8 * 48 * dt, 8 * 48 * dt) - v2df(7 * 48 * dt, 7 * 48 * dt) * RRc * _R2c;
    v2df Ba = v2df(5 * 24 * dt, 5 * 24 * dt) - v2df(4 * 24 * dt, 4 * 24 * dt) * RRa * _R2a;
    v2df Bc = v2df(5 * 24 * dt, 5 * 24 * dt) - v2df(4 * 24 * dt, 4 * 24 * dt) * RRc * _R2c;
    v2df _R4a = _R2a * _R2a;
    v2df _R4c = _R2c * _R2c;
    v2df _R6a = _R4a * _R2a;
    v2df _R6c = _R4c * _R2c;
    v2df _R8a = _R4a * _R4a;
    v2df _R8c = _R4c * _R4c;
```

そのアセンブラ

```
/*    481 */    sxar2
/*    481 */    fmuld,s %f48,%f48,%f60
/*    481 */    fsubd,s %f62,%f42,%f62

/*    481 */    sxar2
/*    481 */    ldd     [%o4+%l1],%f324
/*    481 */    ldd     [%xg30+%xg11],%f72

/*    481 */    sxar2
/*    481 */    ldd     [%g1+%xg11],%f328
/*    481 */    fsubd,s %f64,%f42,%f64

/*    481 */    sxar2
/*    481 */    ldsw    [%xg19+40],%xg3
/*    481 */    fmaddd,s        %f44,%f44,%f56,%f56

/*    481 */    sxar2
/*    481 */    fsubd,s %f66,%f46,%f66
/*    481 */    ldd     [%g3+%xg11],%f76

/*    481 */    sxar2
/*    481 */    ldd     [%o4+%xg11],%f332
/*    481 */    ldsw    [%xg19+44],%xg5

/*    481 */    sxar2
/*    481 */    fsubd,s %f68,%f46,%f68
/*    481 */    ldd     [%o5+%l1],%f104

/*    481 */    sxar2
/*    481 */    fmaddd,s        %f50,%f50,%f60,%f60
/*    481 */    fmuld,s %f62,%f62,%f70
```

コンパイラの出力した情報
コンパイラが吐いたアセンブラ
プロファイラの情報

これらをにらみながら地道にチューニングを重ねる日々・・・

※ 興味のある方は → http://mdacp.sourceforge.net/

並列プログラミング

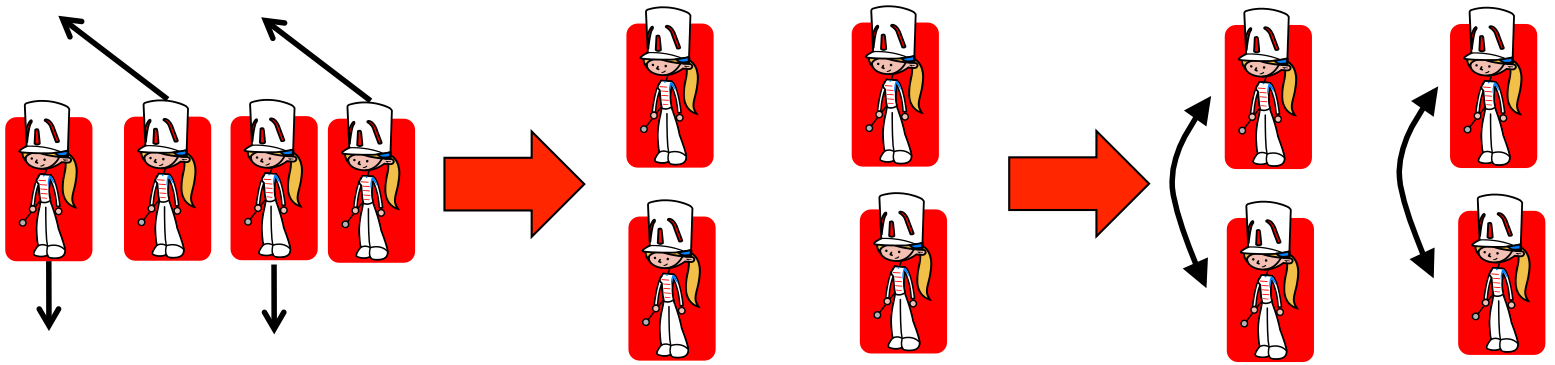
並列プログラミングとは

一つのプログラムで複数のCPUコアを協調動作させること
＝マーチングバンドにおけるコンテ(動作指示書)のようなもの

それぞれの人(プロセス)には通し番号(ランク)が振ってある

指示1: 奇数番の人は左に、奇数番の人は右前に移動せよ

指示2:　隣の人とバトンを交換せよ



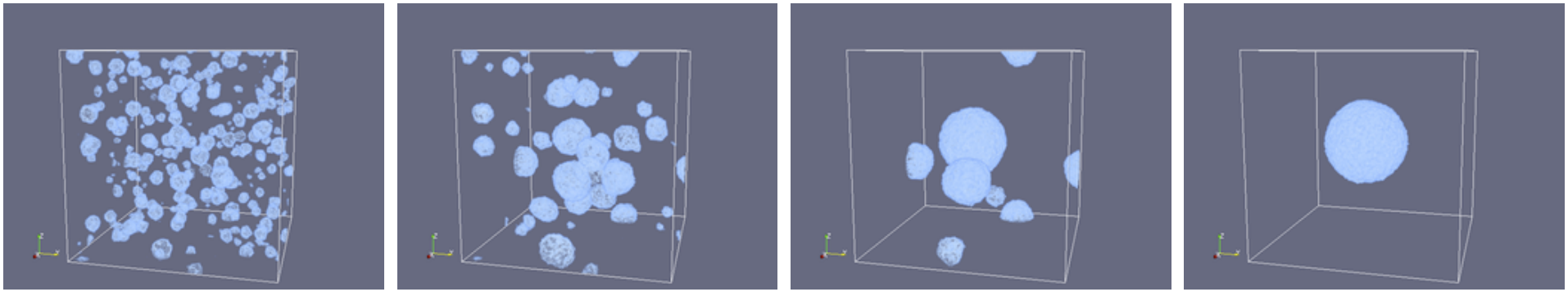
※ 誰が(プロセス)がどこで何をやっているかを常に把握しておく必要がある
※ デバッグが極めて面倒

通信部分のコード例

```
MPI_Sendrecv(&send_buffer[0], send_number, MPI_INT, dest_rank, 0, &recv_buffer[0],
recv_number, MPI_INT, src_rank, 0, MPI_COMM_WORLD, &st);
```

「京」を用いた気泡生成シミュレーション

初期ステージ　　中間ステージ　　最終ステージ



多重核生成　　気泡間相互作用による「つぶしあい」　　単一気泡へ収束

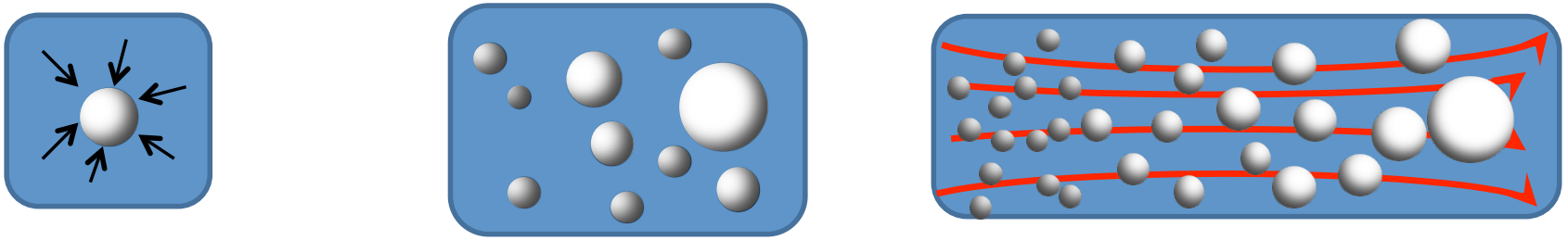
計算規模: 10億粒子の計算。4096ノード×24時間×10回 = 100万ノード時間

物理現象とスケール

イマココ

Movie

ミクロ → マクロ



気泡生成
10 nm
百万粒子

多重気泡生成
100 nm
十億粒子

気泡流
1 um
一兆粒子

The University of Tokyo

世界一を目指すということ

ハードとしての世界一

TOP500における1位の賞状
(理研次世代スーパーコンピュータ開発実施本部より)

ソフトとしての世界一

SC12におけるGordon Bell prize授賞式の様子
マイナビニュースより (http://news.mynavi.jp/photo/articles/2012/11/20/sc12_gordon_bell/images/005l.jpg)

世界一のマシンを使っても、世界一の結果を出すためには
世界一の「何か」が必要　(努力？才能？)

「2位じゃダメなんでしょうか？」にこたえるために・・・

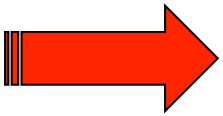
まとめ

・スパコン向けのプログラミングはそんなに難しくない
→ただし、とっても面倒くさい

・スパコンプログラマ≠すごいプログラマ
→ただし、とても人間とは思えないプログラマも存在する

・マイナーな分野ならわりとすぐに世界レベルにはなれる
→ただし「世界一」になるのは極めて大変

・スパコンではハードの進化とソフトの進化の両輪が必要
→ どうやってスパコンプログラマを育成するのか？



Toward Exascale computers...